OPERADORES PARA COMPOSIÇÃO PARALELA: COMPOSIÇÃO GENERALIZADA E ENTRELAÇAMENTO

Alexandre Mota & Augusto Sampaio

Operadores de Composição Paralela

- CSP oferece várias alternativas variando as condições para interação:
  - Composição paralela síncrona

    `P || Q`
  - Composição paralela alfabetizada

    `P [X||Y] Q`
  - Composição paralela generalizada

    `P [|X|] Q`
  - Entrelaçamento

    `P ||| Q`

Entrelaçamento (*Interleaving*)

- Operador para construir processos:

  `P ||| Q`

  dado processos `P` e `Q`
- `P` e `Q` são executados em paralelo, de forma independente, sem sincronizar eventos
- Útil para especificar a composição paralela de *clientes* em um sistema cliente-servidor

Entrelaçamento

- `P ||| Q`
  - oferece os eventos iniciais tanto de `P` quanto de `Q`, e espera até que haja uma comunicação
  - depois da comunicação de um evento a de `P` (`Q`) comporta-se como `P' ||| Q` (`P ||| Q'`), onde `P'(Q')` comporta-se como `P` (`Q`) após a comunicação de a
  - a escolha da execução do mesmo evento de `P` e `Q` é não-determinística

Exemplo

```
ATM1 = incard?c -> pin.fpin(c) ->
       req?n -> dispense!n ->
       outcard.c -> ATM1

CUST1(card) =
  incard.card -> pin?p:S(card) ->
  req.50 ->
  dispense?x:{y | y <- WA, y >=50} ->
  outcard.card -> CUST1(card)
```

Exemplo

```
CUSTOMERS =
   CUST1(1) ||| CUST1(2) |||
   CUST1(3) ||| ...
```

- A composição paralela de ATM1 com CUSTOMERS usa um outro operador que permite sincronização, como ilustrado posteriormente

Processos Paralelos e Sequenciais

Step law

```
   P = c?x:A -> P'
   Q = c?x:B -> Q'

   P ||| Q
=
   c?x:A∪B ->
      if (x∈A∩B) then
         (P' ||| Q) |~| (P ||| Q')
      else if (x∈A) then
              (P' ||| Q)
           else (P ||| Q')
```

Traces

- traces(P ||| Q) =
  $\bigcup$ { s ||| t |
  s ∈ traces(P) ∧
  t ∈ traces(Q) }
- s || t denota os entrelaçamentos dos traces s e t
  - <> ||| s = {s}
  - s ||| <> = {s}
  - <a> ^ s ||| <b> ^ t = {<a>^u | u ∈ s || <b>^t}
    ∪ {<b>^u | u ∈ <a>^s || t}

Leis algébricas

- P ||| Q = Q ||| P
- P ||| (Q ||| R) = (P ||| Q) ||| R
- P ||| (Q |~| R)
  =
  (P ||| Q) |~| (P ||| R)
- P ||| SKIP = P
  - Note particularmente que

    STOP ||| SKIP = STOP

Exercícios

- Verifique intuitivamente se
  - STOP ||| STOP = STOP
  - a -> STOP ||| STOP = a -> STOP
  - P ||| STOP = P
- Usando a Step law prove que

  (a -> STOP ||| b -> STOP)

=

  (a –> b-> STOP)[] (b -> a -> STOP)
- Verifique intuitivamente se

  P ||| (Q [] R)

=

  (P ||| Q) [] (P ||| R)

Resposta dos Exercícios

- Verifique intuitivamente se
  - STOP ||| STOP = STOP - sim
  - a -> STOP ||| STOP = a -> STOP - sim
  - P ||| STOP = P - não
- Usando a Step law prove que
  (a -> STOP ||| b -> STOP)
  =
  (a –> b-> STOP)[] (b -> a -> STOP) – prova simples!
- Verifique intuitivamente se

  P ||| (Q [] R) = (P ||| Q) [] (P ||| R)

  - Não, o lado direito pode gerar não determinismo interno, devido às ocorrências duplas de P. Na verdade, o lado esquerdo é um refinamento do direito

Composição Paralela Generalizada

- Operador para construir processos:

  `P [|X|] Q`

  dado processos `P` e `Q` e um conjunto de eventos `X`

- `P` e `Q` são executados em paralelo mas só sincronizando os eventos em `X`:
  - os outros eventos são realizados de forma independente (como no caso de *Interleaving*)

Exemplo

```
ATM1 = incard?c -> pin.fpin(c) ->
       req?n -> dispense!n ->
       outcard.c -> ATM1

CUST1(card) =
  incard.card -> pin?p:S(card) ->
  req.50 ->
  dispense?x:{y | y <- WA, y >=50} ->
  outcard.card -> CUST1(card)
```

Exemplo

```
CUSTOMERS =
   CUST1(1) ||| CUST1(2) |||
   CUST1(3) ||| ...

ATM1andCUSTOMERS =
    ATM1 [|Events|] CUSTOMERS
```

Composição Paralela Generalizada

- P [|X|] Q
  - sendo A e B respectivamente os eventos iniciais de P e Q, oferece inicialmente o seguinte conjunto de eventos:
    - C =(A\X) U
      (B\X) U
      (A∩B∩X)

Processos Paralelos e Sequenciais

```
P = c?x:A -> P'
Q = c?x:B -> Q'

P [|X|] Q
=
c?x:C ->
 if (x∈X) then P'[|X|] Q'
 else if (x∈A∩B) then
          (P' [|X|] Q)
      |~| (P [|X|] Q')
 else if (x∈A) then
          (P' [|X|] Q)
 else (P [|X|] Q')
```

Traces

- $traces(P \ [|X|] \ Q) =$
  $\bigcup \{ s \ [|X|] \ t \mid s \in traces(P) \wedge t \in traces(Q) \}$
- $s \ [|X|] \ t$ denota as possíveis combinações dos traces s e t

Leis algébricas

- P [|X|] Q = Q [|X|] P
- P [|X|] (Q |~| R)
  =
  (P [|X|] Q) |~| (P [|X|] R)
- P [|X|] (Q [|X|] R)
  =
  (P [|X|] Q) [|X|] R)
  - Esta lei vale apenas se o conjunto de sincronização for o mesmo!

Exercícios

- Quais os processos equivalentes a:
  - `STOP [|X|] STOP = ?`
  - `SKIP [|X|] SKIP = ?`
  - `a -> STOP [|{a}|] STOP = ?`
  - `a -> STOP [|{b}|] STOP = ?`
  - `(?x : A -> P) [|X|] STOP = ?`
  - `a -> STOP [|{a}|] SKIP = ?`
  - `(?x : A -> P) [|X|] SKIP = ?`
  - `(a -> STOP [|{a,b}|] b -> STOP) = ?`
  - `(a -> STOP [|{b}|] b -> STOP) = ?`
- Verifique intuitivamente se

```
  P [|X|] (Q [] R)
=
 (P [|X|] Q) [] (P [|X|] R)
```

Resposta dos exercícios

- Quais os processos equivalentes a:
  - STOP [|X|] STOP = STOP
  - SKIP [|X|] SKIP = SKIP
  - a -> STOP [|{a}|] STOP = STOP
  - a -> STOP [|{b}|] STOP = a -> STOP
  - (?x : A -> P) [|X|] STOP = ?x : A\X -> (P [|X|] STOP)
  - a -> STOP [|{a}|] SKIP = STOP
  - (?x : A -> P) [|X|] SKIP = ?x : A\X -> (P [|X|] SKIP) (ver Lei 6.15)
  - (a -> STOP [|{a,b}|] b -> STOP) = STOP
  - (a -> STOP [|{b}|] b -> STOP) = a -> STOP
- Verifique intuitivamente se

  P [|X|] (Q [] R)

  =

  (P [|X|] Q) [] (P [|X|] R) –

- Falso. Ver considerações no execício anterior.

Restringindo Processos

- Em especificações, os operadores de composição paralela podem ser usados para restringir o comportamento dos processos:
  - `P [|X|] STOP` comporta-se como `P` exceto pela proibição da realização dos eventos em `X`
  - `P [|X|] Q`, onde `Q` só realiza os eventos em `X`, restringe o comportamento de `P`

Relação entre operadores para composição paralela

- Todos os outros operadores podem ser definidos em termos de composição paralela generalizada
  - `Entrelaçamento`
    - `P ||| Q = P [|{}|] Q`
  - `Composição paralela síncrona`
    - `P || Q = P [|Events|] Q`

Relação entre operadores para composição paralela

- Composição paralela alfabetizada
  - `P [X||Y] Q`
    - `Sincronizam em X ∩ Y`
    - `P se comporta independentemente em X\Y e Q em Y\X`
    - `P é bloqueado em Events\X e Q em Events\Y`
    - ```
      P [X||Y] Q =
        (P [|Events\X|] STOP)
      [|X∩Y|]
        (Q [|Events\Y|] STOP)
      ```
      ```

Exercício

- Especifique uma aplicação simples envolvendo um processo buffer e um controlador. O buffer deve ser implementado de forma distribuída, com um processo implementando cada célula do buffer e eventos para obter o valor corrente da célula (*get*) e escrever no valor da célula (*set*). O controlador deve receber requisições de input e output do ambiente, para ler e escrever no buffer, seguindo a política de uma fila circular. Output so pode ocorrer se o buffer não estiver vazio. Input so pode ocorrer se o buffer não estiver cheio.
- Dica: o controlador deve guardar a quantidade atual de elementos no buffer, o id da célula inicial, o ida da célula final e uma *cache* com o elemento mais antigo incluído no buffer.

Resposta do exercício

```
MAX = 5
NUMBERS = {0..MAX}
channel get,set : NUMBERS.NUMBERS

 Celula(i,v) = get.i!v -> Celula(i,v)
              []  set.i?v2 -> Celula(i,v2)
Max_Buffer = 3
Buffer = ||| i : {0..Max_Buffer} @ Celula(i,0)

assert Buffer :[deterministic [FD] ]
assert Buffer :[deadlock free [F] ]
assert Buffer :[livelock free [FD] ]
```

Resposta do exercício

```
channel input,output : NUMBERS
incModulo(x,max) = if (x == max) then 1 else x + 1
Controlador(inf,sup,quant,cache) =
   (quant < Max_Buffer & input?x -> set.inf.x ->
       if (quant == 0) then
         Controlador(inf, incModulo(sup,Max_Buffer), quant+1, x)
       else
         Controlador(inf,incModulo(sup,Max_Buffer), quant+1, cache))
[]
    (quant > 0 & output!cache ->
       if (quant == 1) then
              Controlador(incModulo(inf, Max_Buffer), sup, quant-1, cache)
        else
              get!incModulo(inf,Max)?newCache ->
              Controlador(incModulo(inf, Max_Buffer), sup, quant-1, Newcache))
```

Resposta do exercício

System_Buffer =

(Buffer [|{|get,set|}|] Controlador(0,0,0,0)) \ {|get,set|}

assert System_Buffer :[deterministic [FD]]

assert System_Buffer :[deadlock free [F]]

assert System_Buffer :[livelock free [FD]]

Exercícios

- Do livro texto
  - Essenciais: 2.3.4, 2.5.1, 2.5.2
  - Opcionais: 2.3.1, 2.5.3